Docket No. RSW920000175US1

METHOD AND APPARATUS FOR PROCESSING REQUESTS IN A NETWORK DATA PROCESSING SYSTEM BASED ON A TRUST ASSOCIATION BETWEEN SERVERS

1. Field of the Invention:

The present invention relates generally to an improved data processing system, and in particular to a method and apparatus for processing requests. Still more particularly, the present invention provides a method and apparatus for authenticating the users making the requests and providing a trusted association between servers that handle the requests.

2. Background of the Invention:

The Internet, also referred to as an "internetwork", is a set of computer networks, possibly dissimilar, joined together by means of gateways that handle data transfer and the conversion of messages from the sending network to the protocols used by the receiving network (with packets if necessary). When capitalized, the term "Internet" refers to the collection of networks and gateways that use the TCP/IP suite of protocols.

The Internet has become a cultural fixture as a source of both information and entertainment. Many businesses are creating Internet sites as an integral part of their marketing efforts, informing consumers of the products or services offered by the business or providing other information seeking to engender brand loyalty. Many federal, state, and local government agencies are also employing Internet sites for informational purposes, particularly agencies, which must interact with virtually all segments of society such as the Internal Revenue Service and secretaries of state. Providing informational guides and/or searchable databases of online public records may reduce operating costs. Further, the Internet is becoming increasingly popular as a medium for commercial transactions.

Currently, the most commonly employed method of transferring data over the Internet is to employ the World Wide Web environment, also called simply "the Web". Other Internet resources exist for transferring information, such as File Transfer Protocol (FTP) and Gopher, but have not achieved the popularity of the Web. In the Web environment, servers and clients effect data transaction using the Hypertext Transfer Protocol (HTTP), a known protocol for handling the transfer of various data files (e.g., text, still graphic images, audio, motion video, etc.). The information in various data files is formatted for presentation to a user by a standard page description language, the Hypertext Markup Language (HTML). In addition to basic presentation formatting, HTML allows developers to specify "links" to other Web resources identified by a Uniform Resource Locator (URL). A URL is a special syntax identifier defining a communications path to specific information. Each logical block of information accessible to a client, called a "page" or a "Web page", is identified by a URL. The URL provides a universal, consistent method for finding and accessing this information, not necessarily for the user, but mostly for the user's Web "browser". A browser is a program capable of submitting a request for information identified by an identifier, such as, for example, a URL. A user may enter a domain name through a graphical user interface (GUI) for the browser to access a source of content. The domain name is automatically converted to the Internet Protocol (IP) address by a domain name system (DNS), which is a service that translates the symbolic name entered by the user into an IP address by looking up the domain name in a database.

The Internet also is widely used to transfer applications to users using browsers. With respect to commerce on the Web, individual consumers and business use the Web to purchase various goods and services. In offering goods and services, some companies offer goods and services solely on the Web while others use the Web to extend their reach.

Users exploring the Web have discovered that the content supported by HTML document format on the Web was too limited. Users desire an ability to access applications and programs, but applications were targeted towards specific types of platforms. As a result, not everyone could access applications or programs. This deficiency has been minimized though the introduction and use of programs known as "applets", which may be embedded as objects in HTML documents on the Web. Applets are Java programs that may be transparently downloaded into a browser supporting Java along with HTML pages in which they appear. These Java programs are network and platform independent. Applets run the same way regardless of where they originate or what data processing system onto which they are loaded.

Through applets and Web pages, users generate requests to access resources on the Web. Reverse Proxy servers may be used to act as a gateway into an Intranet environment. The users trying to access a resource will make requests to a reverse proxy server which would forward the request to a backend server that processes that particular requests. It is common for using these reverse proxy servers for authentication purposes as well. In other words, these servers make sure that users are who they say they are in a request This authentication may take many different forms, including the use of user IDs and passwords. Before forwarding this request to the backend server, the reverse proxy server might include and/or modify information on the authenticated user's identity. This may be in the form of a header, a credential token or in some other authentication data format. Thereafter, authenticated requests are sent to backend services for processing. The present invention recognizes that most backend services do not understand or recognize credential information that might be passed along with or within a request from a reverse proxy server. It also recognizes the fact that the backend server must trust the reverse proxy server in order to accept and work with the forwarded request.

Therefore, it would be advantageous to have an improved method and apparatus for handling authentication of requests between different servers that have established a trust relationship.

SUMMARY OF THE INVENTION

The present invention provides a method, apparatus, and computer implemented instructions for handling requests in a network data processing system. The network data processing system includes a network and clients connected to the network. A first server is present in which the first server receives a request from a client to access a resource, performs an authentication process with the client, adds and/or modifies information in the request in which the information indicates that the request is from a trusted source to form a modified request, and sends the modified request for processing. This modified request is received by a second server. This second server determines whether the first server is a trusted server based on the information, and provides access to the resource in response to a determination that the first server is a trusted server, the trusted server has already authenticated the end user who made this request and the end user is authorized to the requested resource.

BRIEF DESCRIPTION OF THE DRAWINGS

The novel features believed characteristic of the invention are set forth in the appended claims. The invention itself, however, as well as a preferred mode of use, further objectives and advantages thereof, will best be understood by reference to the following detailed description of an illustrative embodiment when read in conjunction with the accompanying drawings, wherein:

Figure 1 is a pictorial representation of a network of data processing systems in which the present invention may be implemented;

Figure 2 is a block diagram of a data processing system that may be implemented as a server, such as server **104** in **Figure 1**, in accordance with a preferred embodiment of the present invention;

Figure 3 is a diagram illustrating components used in forming authentication of requests in accordance with a preferred embodiment of the present invention;

Figures 4A and **4B** are diagrams illustrating properties files used for recognizing trusted servers in accordance with a preferred embodiment of the present invention;

Figure 5 is a flowchart of a process used for handling a user request at reverse proxy security server in accordance with a preferred embodiment of the present invention;

Figure 6 is a flowchart of a process used for handling requests forwarded from a reverse proxy security server in accordance with a preferred embodiment of the present invention; and

Figure 7 is a diagram of code used in an interceptor in accordance with a preferred embodiment of the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

With reference now to the figures, **Figure 1** depicts a pictorial representation of a network of data processing systems in which the present invention may be implemented. Network data processing system **100** is a network of computers in which the present invention may be implemented. Network data processing system **100** contains a network **102**, which is the medium used to provide communications links between various devices and computers connected together within network data processing system **100**. Network **102** may include connections, such as wire, wireless communication links, or fiber optic cables.

In the depicted example, a server **104** is connected to network **102** along with storage unit **106**. In addition, clients **108**, **110**, and **112** also are connected to network **102**. These clients **108**, **110**, and **112** may be, for example, personal computers or network computers. Additionally, server **114** is connected to server **104** via network server **116**. In the depicted examples, server **104** is a reverse proxy security server, while server **114** is a backend server. A proxy server is also called a "proxy" or "application level gateway". A proxy server is an application that breaks the connection between sender and receiver. All input is forwarded out a different port, closing a straight path between two networks and preventing a hacker from obtaining internal addresses and details of a private network.

Proxy servers are for requests going "out" from a computer and simulates to the end server that it is coming from an entity (does not disclose the requesting machine). A reverse proxy server is the opposite - it hides the servers that serve the request but not the clients that request the resource. Proxy servers are available for common Internet services; for example, an HTTP proxy is used for Web access, and an SMTP proxy is used for e-mail. Proxies generally employ network address translation (NAT), which presents one organization-wide IP address to the Internet. It funnels all user requests to the Internet and fans responses back out to the appropriate users. Proxies may also cache Web pages, so that the next request can be obtained locally. In the depicted examples, server **104** is a reverse proxy security server, which is used to authenticate users requesting access to resources or services provided by server **114**.

In the depicted example, server **114** provides data, such as boot files, operating system images, and applications to clients **108-112**. This access is provided once users requesting these resources are authenticated by server **104**. If a user is authenticated by server **104**, server **104** will include or associate additional information with the request and pass that request to server **114**. Server **114** will recognize server **104** as a trusted server based on the added information.

Network data processing system **100** may include additional servers, clients, and other devices not shown. In the depicted example, network data processing system **100** is the Internet with network **102** representing a worldwide collection of networks and gateways that use the TCP/IP suite of protocols to communicate with one another. At the heart of the Internet is a backbone of high-speed data communication lines between major nodes or host computers, consisting of thousands of commercial, government, educational and other computer systems that route data and messages. Of course, network data processing system **100** also may be implemented as a number of different types of networks, such as for example, an intranet, a local area network (LAN), or a wide area network (WAN). **Figure 1** is intended as an example, and not as an architectural limitation for the present invention.

Referring to **Figure 2**, a block diagram of a data processing system that may be implemented as a server, such as server **104** in **Figure 1**, is depicted in accordance with a preferred embodiment of the present invention. Data processing system **200** may be a symmetric multiprocessor (SMP) system including a plurality of processors **202** and **204** connected to system bus **206**. Alternatively, a single processor system may be employed. Also connected to system bus **206** is memory controller/cache **208**, which provides an interface to local memory **209**. I/O bus bridge **210** is connected to system bus **206** and provides an interface to I/O bus **212**. Memory controller/cache **208** and I/O bus bridge **210** may be integrated as depicted.

Peripheral component interconnect (PCI) bus bridge **214** connected to I/O bus **212** provides an interface to PCI local bus **216**. A number of modems may be connected to PCI bus **216**. Typical PCI bus implementations will support four PCI expansion slots or add-in connectors. Communications links to network computers **108-112** in **Figure 1** may be provided through modem **218** and network adapter **220** connected to PCI local bus **216** through add-in boards.

Additional PCI bus bridges **222** and **224** provide interfaces for additional PCI buses **226** and **228**, from which additional modems or network adapters may be supported. In this manner, data processing system **200** allows connections to multiple network computers. A memory-mapped graphics adapter **230** and hard disk **232** may also be connected to I/O bus **212** as depicted, either directly or indirectly.

Those of ordinary skill in the art will appreciate that the hardware depicted in **Figure 2** may vary. For example, other peripheral devices, such as optical disk drives and the like, also may be used in addition to or in place of the hardware depicted. The depicted example is not meant to imply architectural limitations with respect to the present invention.

The data processing system depicted in **Figure 2** may be, for example, an IBM RISC/System 6000 system, a product of International Business Machines Corporation in Armonk, New York, running the Advanced Interactive Executive (AIX) operating system.

The present invention provides a method, apparatus, and computer implemented instructions for authenticating and handling requests within a network data processing system. The present invention recognizes that, in presently available systems, backend servers are unable to understand the format of credential information that may be passed to them by a reverse proxy security server. More specifically, a reverse proxy security server performs authentication on a coarse granularity based on a first invocation of a server, such as a Web application server, residing behind the reverse proxy security server. After initial security checks, the reverse proxy security server forwards the request to the Web application server along with credential information about the user generating the request. In these examples, this information is included within the request itself. The present invention provides a mechanism for a Web application server to authenticate or establish a trust relationship with the reverse proxy security server forwarding the request.

Further, the mechanism of the present invention allows a Web application server to understand the format of the information added to the request by the reverse proxy security server. In these examples, the mechanism involves the use of a trust association interceptor placed on the Web application server. An interceptor is present for each type of reverse proxy security server to handle requests and determine whether a trust relationship exists, as well as to understand the format of the information provided by the reverse proxy security server.

Turning next to **Figure 3**, a diagram illustrating components used in forming authentication of requests is depicted in accordance with a preferred embodiment of the present invention. In this example, a user at client **300** generates a request to a servlet, which is received by reverse proxy security server **302**. For example, the user may enter a URL in a Web browser, which results in a generation of the request. Reverse proxy security server **302** will consult a configuration database and determine whether the requested URL is protected. If this URL is a protected URL, then an authentication process is initiated. Authentication of the user is performed by reverse proxy security server **302** through an issuance of an authentication challenge to the user client **300**. This challenge may, for example, request a user ID and password or a client certificate. A response is returned to reverse proxy security server **302** by client **300**.

If authentication of the user is successful, then reverse proxy security server **302** adds credential information to the request and passes the request to Web application server **304** for processing. In the depicted examples, information is placed into the header of the request. For example, a value may be included to represent a valid name in a database maintained by Web application server **304**. In these examples, the requests are HTTP requests and the information is placed into a HTTP header. When the request is received by Web application server **304**, this server will determine whether the requested resource is a protected one. In the depicted examples, the determination is provided by Web server plug-in **306**, which manages resources in Web application server **304**. If the resource is a protected resource, Web application server **304** is configured to recognize that the request is from a trusted server, such as reverse proxy security server **302**. Security application **308** determines whether the resource is a protected one. Security collaborator **310** is configured to recognize that a reverse proxy security server authentication service is present as a front end to Web application server **304**.

The identification of this trust relationship is implemented using trust association interceptors **312**, **314**, and **316** in these examples. A trust association interceptor is present for every type of reverse proxy security server that acts as a front end to Web application server **304**. Each one of the interceptors is presented with the request in these examples. If the request is accepted to be handled by an interceptor, the interceptor is asked to validate the trust relationship. If that succeeds, a trust relationship is then recognized as being present. That interceptor also retrieves user information present within the headers of the forwarded request. The retrieved information is validated. In other words, the credential passed in by the reverse proxy security server is analyzed to obtain the user information from the credential information and a determination is made as to whether the user information is present within a database of user information in the web application server.

Upon successful validation of the user, this information is then used to determine whether the user is authorized to access the resource in the request. For example, a user name may be compared against a user database or registry to determine whether the user is allowed to access the resource. In this example, the determination is made by security collaborator **310,** consulting with security application **308**. If the user is authorized to access the resource, then a security context with the user's credential information is generated and passed on to servlet engine **318**.

The request is passed to servlet engine **318** regardless of the result of the authorization check described above. In this example, servlet engine **318** is a hit count servlet used to count the number of requests for a URL. Servlet **318** will invoke a method on a Java bean **320**. In the depicted examples, a method call from servlet **318** to Java bean **320** causes a security check to be performed by security collaborator **310** to determine whether the user is authorized to invoke the method on Java bean **320**. If the user is authorized to invoke the method, then the method is executed and results are returned to the user at client **300**.

In these examples, Web server plug-in **306**, security application **308**, security collaboration **310**, and trust association interceptors **312**-**316** are components within Web application server **304**.

Turning next to **Figures 4A** and **4B**, diagrams illustrating an example of how the system can be configured using a set of property files. The properties files used for recognizing trusted servers are depicted in accordance with a preferred embodiment of the present invention. In **Figure 4A**, property file **400** illustrates how to enable trust association between the Web Application Server and the reverse proxy servers, as shown in line **402**. In line **404**, the types of reverse proxy servers that are currently available as a front end to the Web Application Server can be specified. In the example, there is only one reverse proxy server and the type is webseal36. Every reverse proxy server type must have a corresponding interceptor. In line **406**, the interceptor for webseal36 is implemented by the specified Java class. Optionally, every interceptor may have its own configuration or property file. In line **408**, the name of the property file for the webseal36 interceptor type is specified.

In **Figure 4B**, property file **410** illustrates an exemplary property file of a given interceptor. The file defines information that will be provided by the interceptor's corresponding reverse proxy security server. In this example, line **412** specifies the string "iv-creds" as the specific id information that will be added by the proxy server to the request to signify to the interceptor that the request has been authenticated by that proxy server.

Turning now to **Figure 5**, a flowchart of a process used for handling a user request at reverse proxy security server is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 5** may be implemented in a reverse proxy security server, such as server **104** in **Figure 1**.

The process begins by receiving a request from a user at a client (step **500**). In this example, this request may be initiated by a user entering or selecting a URL. Authentication is then performed on the request, as needed (step **502**). The process may identify whether the URL is protected. The authentication step includes issuing a challenge and receiving a response from the user. The challenge may be, for example, a request for a user ID and password.

A determination is made as to whether the user has been authenticated (step **504**) or not. If the user is authenticated, then one or more selected fields or values are added or modified to the request (step **506**). This new information may include, for example, user identification information and an identification of the reverse proxy security server. The request is then sent to the backend server (step **508**) with the process terminating thereafter.

Turning back to step **504**, if the user is not authenticated, an error is returned (step **510**) with the process terminating thereafter. In this example, the error is returned to the user, indicating that authentication has not occurred. Alternatively, the user may be prompted to retry authentication, depending on the particular implementation.

Turning next to **Figure 6**, a flowchart of a process used for handling requests forwarded from a reverse proxy security server is depicted in accordance with a preferred embodiment of the present invention. The process illustrated in **Figure 6** may be implemented in a server, such as, server **114** in **Figure 1**. More specifically, the process illustrated may be implemented in a security collaborator together with an interceptor on the server.

The process begins with the security collaborator receiving a request (step **600**). The request is sent to each available interceptor. Next, the security collaborator inquires each available interceptor if it can process the request (step **602**). Each interceptor determines whether it is able to process the request and returns the result to security collaborator. A determination is made by the Security Collaborator as to whether any of the available interceptors responded positively to the inquiry (step **604**).

If an interceptor is available to process the request, that interceptor is asked to validate trust with the proxy server (step **606**). This helps the backend server to support multiple reverse proxy security servers (e.g, different types or different security restrictions depending on the way they are deployed). The selected interceptor might require a user name representing the proxy server and password within the request. This determination may be made by comparing the values to expected values located in a data structure, such as a registry table or database. The presence of a value or field within a header may suffice or a specific value may be expected.

Then, a determination is made as to whether the proxy server is trusted (step **608**). If the proxy server is not trusted, an error is returned (step **610**) with the process terminating thereafter.

If the proxy server is trusted, the username authenticated by the proxy server is retrieved (step **612**). Next, credentials are created from the username (step **614**). A determination is then made as to whether or not the credentials were created successfully (step **616**). If the credentials were not created successfully, an error is returned (step **618**). If the credentials were created successfully, the process proceeds to step **628** to begin authorization as will be described below.

With reference again to step **604**, if it has been determined that no interceptor can process a request, the security collaborator itself will authenticate the request (step **620**). This is for the cases where the request might be directed from a user directly to the backend server instead of going through a reverse proxy security server. A determination is made as to whether the authentication is okay (step **622**). If the authentication is unsuccessful, an error is returned (step **624**) with the process terminating thereafter. If the authentication is successful, user credentials are created (step **626**). Then, a determination is made as to whether the user is authorized using the credentials (step **628**). If the user is authorized using the credentials, the request is allowed (step **630**) with the process terminating thereafter. If the user is not authorized using the credentials, the request is denied (step **632**) with the process terminating thereafter.

Turning now to **Figure 7**, a diagram of application program interfaces (APIs) available in an interceptor are depicted in accordance with a preferred embodiment of the present invention. Code **700** is the set of APIs required upon every interceptor to implement in order to be able to collaborate with a Web Application server such as server **304** in **Figure 3**. More specifically, these API's are called by the Security Collaborator such as **310** in **Figure 3**. Code **700** is written in Java in this example.

Trust association is specific to the characteristics of each type of reverse proxy security server. An interceptor is present for each type of reverse proxy security server. For example, a user ID and password may be present in an HTTP authorization header to validate the credentials of the reverse proxy security server. Code **700** performs validation of the presence of this information, as well as extracting information for further use.

Section **702** is used to determine whether this interceptor is designed to process a particular request. This corresponds to the call made by the Security Collaborator to the interceptor in step **602** of **Figure 6**. Section **704** is used to determine whether the interceptor trusts the server through which the request has been routed. This corresponds to the call made by the Security Collaborator to the interceptor in step **606** of **Figure 6**. If the server fails validation or is unable to provide the required information, an exception is thrown, which causes a denial of access to the requested resource.

Section **706** is used to retrieve the user name of the client originating the request. This corresponds to the call made by the Security Collaborator to the interceptor in step **612** of **Figure 6**. This section is invoked if the validation of the server is successful.

Thus, the present invention provides a method, apparatus, and computer implemented instructions for handling requests received from a proxy server at a backend server. The mechanism involves establishing whether the server forwarding the request is a trusted server. This validation, in these examples, involves determining whether values are present within a request in the expected locations, as well as validating the values. The mechanism also involves extracting user information to further determine whether access to the request resource is authorized.

It is important to note that while the present invention has been described in the context of a fully functioning data processing system, those of ordinary skill in the art will appreciate that the processes of the present invention are capable of being distributed in the form of a computer readable medium of instructions and a variety of forms and that the present invention applies equally regardless of the particular type of signal bearing media actually used to carry out the distribution. Examples of computer readable media include recordable-type media such a floppy disc, a hard disk drive, a RAM, CD-ROMs, and transmission-type media such as digital and analog communications links.

The description of the present invention has been presented for purposes of illustration and description, and is not intended to be exhaustive or limited to the invention in the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art. The embodiment was chosen and described in order to best explain the principles of the invention, the practical application, and to enable others of ordinary skill in the art to understand the invention for various embodiments with various modifications as are suited to the particular use contemplated.